High-level Programming Languages and Systems for Cyber-Physical Systems

Kazunori Ueda
Waseda University, Tokyo, Japan

Halmstad Summer School of Cyber-Physical Systems, July 17-21, 2017
Cyber-physical systems (CPS, 2000’s–) = systems with *computational* and *physical* components

Hybrid systems (1990’s–) = dynamical systems with *continuous* and *discrete* behavior

Various aspects:
- embedded systems, IoT, sensor network, big data, social/network infrastructure, distributed computing, security, ...

Computational foundations for
- interacting with the physical world (= implementing CPSs)
- modeling, simulation and verification
Computing/modeling paradigms for CPSs

◆ Key issue
= modeling of, and interfacing with, the physical world

- Physical systems
  - Continuous (+ discrete) domain
  - Math with differential (+ algebraic) equations
  - Time

- Computer systems
  - Discrete domain
  - Programming languages
  - Algorithms
  - Abstraction

How to reconcile them with computing abstraction of physical systems?
Computing/modeling paradigms for CPSs


NSF Workshop On Cyber-Physical Systems, October, 2006

4. Research directions

- Putting time into programming languages
- Rethinking the OS/programming split
- Rethink the hardware/software split
- Memory hierarchy with predictability
- Memory management with predictability
- Predictable, controllable deep pipelines
- Predictable, controllable, understandable concurrency
- Concurrent components
- Networks with timing
- Computational dynamical systems theory
Hybrid systems

- Systems whose states can make both continuous and discrete changes

Examples:
- bouncing ball, billiard, . . .
- thermostat + air conditioner + room
- traffic signals + roads + cars

In general:

Dynamical systems whose description involves case analysis

- physical, biological, control, cyber-physical, etc.

- Relates to computer science, control engineering and apps.

- Programming language aspects rather unexplored
Challenges and questions

◆ Designing and implementing programming/modeling languages for hybrid systems
  ● What are the basic notions and constructs? cf. automata (concrete) vs. \( \lambda \)-calculus (abstract)
  ● Are they simple and accessible to non-specialists (e.g., engineers outside CS)?

◆ Language constructs are divided into
  ● those determining the underlying computational model (primitives)
  ● those motivated by software engineering point of view (user language)
Modeling frameworks for hybrid systems

◆ **Hybrid Automata** and other “hybrid” models (Petri nets, I/O automata, Process Algebra, etc.)

◆ **Modeling languages and tools** with equations and updates
  - Modelica, Acumen, Ptolemy, Hybrid Language, ...

◆ **Constraint-based** languages and tools (domain = functions over time)
  - iSAT (Boolean+arithmetic constraint solver)
  - Hybrid CC (hybrid concurrent constraint language)
  - CLP(F) (constraint LP over real-valued functions)
  - Kaleidoscope ’90 (discrete time)
  - HydLa (constraint hierarchy)

Constraint Programming (CP)

- A **declarative** programming paradigm in which a problem is described using equations/inequations over continuous or discrete domains

- Variables: \( x_1, ..., x_5 \)
- Domain: \( 1 \leq x_i \leq 5 \)
- Constraints:
  - if \( i \neq j \) then
    - \( x_j \neq x_i \)
    - \( x_j \neq x_i + |j - i| \)
    - \( x_j \neq x_i - |j - i| \)
Constraint Programming (CP)

- **Features and essence**
  - *No algorithms*: CP languages are often called modeling languages
  - Developed in AI and Logic Programming communities
    - where the central interest has been constraint satisfaction and constraint propagation
    - many libraries for mainstream languages
    - CP languages are mostly based on Logic Programming
  - Another view of CP: *computing with partial information*
    - by means of symbolic execution
Different flavors and applications
- Constraint satisfaction problems (CSPs)
  - Domains: finite, real, interval, ...
- SMT (satisfiability modulo theories)
  - complex combination of logical connectives
  - usually not compute most general solutions
- (Constraint-based) Concurrency
  (a.k.a. Concurrent Constraint Programming)

Communication: **tell**ing and **ask**ing of constraints
Synchronization:ugnt (also for conditionals)
Composition: $\wedge$
Hiding: $\exists$ (also for fresh name creation)
Early history of constraint-based concurrency

*Guarded Horn Clauses; not to be confused with Glasgow Haskell Compiler (1990s)

1980
- Relational Language
  - Concurrent Prolog
    - PARLOG

1985
- GHC *
  - FCP
  - Flat GHC
  - PARLOG
    - KL1
    - Strand
      - ALPS
      - CCP
  - Moded Flat GHC
    - PCN
    - JANUS
    - CC++
      - Oz/Mozart

1990
- P-Prolog
  - Andorra Prolog
  - AKL

Single Idea: Dataflow Synchronization

Originated by process interpretation of logic programs

factorial(X,Y) :- X=:=0 | Y:=1.
factorial(X,Y) :- X > 0 | X1:=X−1, factorial(X1,Y1), Y:=X*Y1.

constraints on immutable variables

M=5
N=120

factorial(M,N)
Constraint-based concurrency

- Inverter accepting a sequence of input data

\[
\ldots 1 \ 0 \ 1 \ 1 \ 0 \quad \ldots 0 \ 1 \ 1 \ 0 \ 1
\]

\text{nots}

\[
\text{nots}([], \ Y) : - \text{true} \mid Y=[].
\]

\[
\text{nots}([0|X],Y0) : - \text{true} \mid Y0=[1|Y], \text{nots}(X,Y).
\]

\[
\text{nots}([1|X],Y0) : - \text{true} \mid Y0=[0|Y], \text{nots}(X,Y).
\]

- Discrete event systems can be represented using possibly infinite lists.
  - e.g., \([0,1,1,0,1|A]\)
Constraints imposed by “nots(X,Y)”: 

<table>
<thead>
<tr>
<th>Observed</th>
<th>Published</th>
<th>Rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>X=[0,1,1,0,1]</td>
<td>Y=[1,0,0,1,0]</td>
<td>(none)</td>
</tr>
<tr>
<td>X=[]</td>
<td>Y=[]</td>
<td>(none)</td>
</tr>
<tr>
<td>X=[0,1,1,0,1</td>
<td>X’]</td>
<td>Y=[1,0,0,1,0</td>
</tr>
<tr>
<td>(none)</td>
<td>(suspending)</td>
<td>nots(X,Y)</td>
</tr>
<tr>
<td>X=[2</td>
<td>_]</td>
<td></td>
</tr>
<tr>
<td>X=[0</td>
<td>_], Y=[0</td>
<td>_]</td>
</tr>
</tbody>
</table>
Constraint Programming for hybrid systems

- Declarative description of hybrid systems
  = constraint programming of functions over time
  - cf. constraint programming over infinite sequences

- Many features are inherited from constraint-based concurrency
  - Implication (⇒) for synchronization and conditionals
  - Conjunction (∧) for parallel composition
  - Existential quantification (∃) for hiding

\[\square(e\text{-}stop = 1 \implies speed' = -4.0)\]

(ask) (tell)
Challenges from the language perspective

◆ Establish a high-level programming/modeling language
  ● equipped with the notion of *continuous time*,
  ● equipped with the notion of *continuous changes*,
  ● that properly handles *uncertainties* and *errors of real values*,
  ● that properly handles *conditional branch* under uncertainties and errors of real values,
  ● equipped with constructs for *abstraction* and *parallel composition*.
  ● etc.

◆ Establish semantical foundations

◆ Establish implementation technologies
Rigorous simulation

• Computers were born for numerical simulation, and simulation (in a broad sense) is still an important application of high-performance computers for the design and analysis of all kinds of systems.

• “How (much) can we trust these simulation results?”
  
  • For some simple problems, ordinary simulation with a standard tool cannot yield a single significant digit.
Rigorous simulation

- Simulation of hybrid systems is particularly hard and can easily go qualitatively wrong (due to conditional branch). A technique for rigorous simulation is very important.

- Some CPSs are safety-critical or mission-critical also.

Collision of three bodies

Small errors make big differences!

Collision avoidance model
Rigorous simulation vs. verification

- Most research on hybrid systems aims at verification as decision problems
  - yes/no answer (i.e., whether it works)
  - possibly with counterexamples (i.e., why it doesn’t work)

- Rigorous simulation will require less from you and tell you more
  - no proof skills (cf. interactive theorem solving)
  - no proof goals (cf. automatic verifier)
    - still can be used to prove something (e.g., W. Tucker’s proof on Lorenz attractors, R. E. Moore Prize 2002)
  - (often visualized) trajectories (i.e., how it works)
  - error margin (i.e., how safe it is)
The field of hybrid systems comes with many notations, concepts and techniques; rather difficult to get into.

Our challenge is to see whether a rather simplistic formalism can address various aspects of hybrid systems.

Goals:
- Identifying computational mechanisms
- Modeling and understanding systems that are not large but may exhibit problematic behavior

Non-goals (currently):
- Modeling large-scale systems
HydLa: Overview and features (2/4)

- **Declarative** ($\leftrightarrow$ Procedural)
  - Minimizes new concepts and notations by employing popular mathematical and logical notations
    - $=, \leq, +, \times, \frac{d}{dx}, \wedge, \Rightarrow, \Leftrightarrow, \ldots$
  - Describes systems as logical formulae with hierarchy
    - No algorithmic constructs such as states and state changes, iteration, transfer of control, etc.
  - Still, it turns out that the semantics comes with large design space, e.g.,
    - how to compare two uncertain values?
    - what continuity should we assume?
HydLa: Overview and features (3/4)

- Constraint-based
  - Basic idea: defines functions over time using constraints including ODEs, and solves initial value problems
    - cf. streams are defined by difference equations
  - Handles partial (incomplete) information properly
    - Intervals (e.g., $x \in [1.0, 3.5]$) fit well within the constraint-based framework
    - Allows modeling and simulation of parametric hybrid systems
  - Symbolic computation based on consistency checking
    - Powered by numerical techniques
Features constraint hierarchies (Alan Borning, 1992)

Motivation: It’s often difficult to describe systems so that the constraints are consistent and well-defined.

Examples: bouncing ball (, billiard, . . .)

- A ball normally obeys the law of gravity (default), while it obeys the collision equation when it bounces (exception).

- The frame problem (McCarthy and Hayes, 1960s) occurs in the description of complex systems.
  - We can’t enumerate all possible exceptions

- Want to define these properties concisely and in a modular manner.
Example 1: Sawtooth function

\begin{align*}
\text{INIT} & \iff f = 0. \\
\text{INCREASE} & \iff \square (f' = 1). \\
\text{DROP} & \iff \square (f^- = 1 \implies f = 0).
\end{align*}

\text{INIT, } (\text{INCREASE} \ll \text{DROP}).

- Describes properties at time 0.
- Time argument is implicit: \( \square (f' = 1) \) means \( \forall t \geq 0 (f'(t) = 1) \)
- \( f^- \) stands for the left-hand limit of \( f \).
Example 1b: Sawtooth function

- INIT \(\iff 0 \leq f < 1.\)
- INCREASE \(\iff \square(f' = 1).\)
- DROP \(\iff \square(f- = 1 \Rightarrow f = 0).\)

INIT, (INCREASE \(\ll\) DROP).

- Describes properties at time 0.
- Family of sawtooth functions with the slope 1 and the range \([0, 1)\).
- Value of \(f\) at a specific time point is just known to be \([0, 1)\), but all trajectories reach all values in \([0, 1)\) and oscillate.
Example 2: Bouncing ball

\[ \text{INIT} \iff ht = 10 \land ht' = 0. \]

\[ \text{PARAMS} \iff \Box (g = 9.8 \land c = 0.5). \]

\[ \text{FALL} \iff \Box (ht'' = -g). \]

\[ \text{BOUNCE} \iff \Box (ht^- = 0 \Rightarrow ht' = -c \times (ht'-)). \]

\[ \text{INIT, PARAMS, (FALL} \ll \ll \text{BOUNCE)}. \]

- When the ball is not on the ground, \{INIT, PARAMS, FALL, BOUNCE\} is maximally consistent.
- When the ball is on the ground, \{INIT, PARAMS, BOUNCE\} is maximally consistent.
- At each time point, HydLa adopts a maximally consistent set of rules that respects constraint priority.
Demo

- HyLaGI (HydLa Guaranteed Implementation) and webHydLa
  - http://webhydla.ueda.info.waseda.ac.jp/
  - http://www.ueda.info.waseda.ac.jp/hydra/
Constraint hierarchy

- Constraint hierarchy specified by “<<“ determines possible combination of rules

\[
\text{INIT, PARAMS, (FALL << BOUNCE)}
\]

where rules with highest priority are “required” constraints

- Basic HydLa (next slide) considers a partially ordered set of “set of rules” induced from the constraint hierarchy.

\[
\{\text{INIT, PARAMS, FALL, BOUNCE}\}
\]

\[
\{\text{INIT, PARAMS, BOUNCE}\}
\]
### Syntax of Basic HydLa

<table>
<thead>
<tr>
<th>(program)</th>
<th>$P ::= (RS, DS)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(rule sets)</td>
<td>$RS ::= \text{poset of sets of } R$</td>
</tr>
<tr>
<td>(definitions)</td>
<td>$DS ::= \text{set of } D\text{'s with different LHSs}$</td>
</tr>
<tr>
<td>(definition)</td>
<td>$D ::= R \iff C$</td>
</tr>
<tr>
<td>(constraint)</td>
<td>$C ::= A \mid C \land C \mid G \Rightarrow C \mid \Box C \mid \exists x. C$</td>
</tr>
<tr>
<td>(guard)</td>
<td>$G ::= A \mid G \land G$</td>
</tr>
<tr>
<td>(atomic constraint)</td>
<td>$A ::= E \ \text{relop} \ E$</td>
</tr>
<tr>
<td>(expression)</td>
<td>$E ::= \text{ordinary expression} \mid E' \mid E\neg$</td>
</tr>
</tbody>
</table>
A program is a pair of
- partially ordered set of “sets of rules” ($RS$) and
- rule definitions ($DS$).

Example of $RS$:
\[
\{\text{INIT, PARAMS, BOUNCE}\} \prec \{\text{INIT, PARAMS, FALL, BOUNCE}\}
\]
- How to derive $RS$ from $<<$ is beyond Basic HydLa.

HydLa / Basic HydLa is a **language scheme** in which the underlying constraint system is left unspecified.

- $\exists x \cdot C$ realizes dynamic creation of variables.

  Example: creation and activation of new timers
  - $\exists$ is eliminated at runtime using Skolem functions.
Semantics of Basic HydLa

- **Declarative semantics** (Ueda, Hosobe, Ishii, 2011)
  - What trajectories does a HydLa program denote?

- **Operational semantics** (Shibuya, Takata, Ueda, Hosobe, 2011)
  - How to compute the trajectories of a given HydLa program?

- Unlike many other programming languages, declarative semantics was designed first, since
  - completeness of the operational semantics can’t be expected and
  - diverse execution methods are to be explored.
The purpose of a HydLa program is to define the constraints on a family of trajectories.

\[ \overline{x}(t) = \{x_i(t)\}_{i \geq 1} \quad (t \geq 0) \]

Declarative semantics, first attempt

\[ \overline{x}(t) \models (RS, DS) \]

- Works fine for programs not containing \( \square \) in the consequents of conditional constraints \( G \Rightarrow C \) [JSSST ’08].

Example: systems with a fixed number of components and without delays
Declarative semantics of Basic HydLa

- Not only trajectories, but also effective constraint sets defining the trajectories, change over time.
  - Reason 1: Maximally consistent sets may change.
  - Reason 2: Conditional constraints may discharge their consequents.
    - When the consequent of a constraint starts with $\square$, whether it’s in effect or not depends on whether the corresponding guard held in the past.

- Declarative semantics (refined)

\[
\langle \bar{x}, Q \rangle \models (RS, DS)
\]

\[
Q(t) : \text{rule definitions with dynamically added consequents}
\]
◆ We identify a conjunction of constraints with a set of constraints.

◆ We regard a set of constraints as a function over time.
  ● A constraint $C$ in a program is regarded as a function
    $$
    \begin{cases}
    C(0) = C, \\
    C(t) = \{ \} \ (t > 0).
    \end{cases}
    $$

◆ □-closure * : Unfolds (or unboxes) the topmost □-formulas dynamically and recursively.

Example: $C = \{ f=0, □\{ f'=1 \} \}$

$$
\begin{cases}
C^*(0) = \{ f=0, f'=1, □\{ f'=1 \} \} \\
C^*(t) = \{ f'=1 \} \ (t > 0)
\end{cases}
$$
<table>
<thead>
<tr>
<th>Declarative semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle \bar{x}, Q \rangle \vDash (RS, DS) \iff (i) \land (ii) \land (iii) \land (iv)$, where</td>
</tr>
<tr>
<td>(i) $\forall t \forall R (Q(R)(t) = Q(R)^*(t))$</td>
</tr>
<tr>
<td>(ii) $\forall t \forall R (DS^<em>(R)(t) \subseteq Q(R)^</em>(t))$</td>
</tr>
<tr>
<td>(iii) $\forall t \exists E \in RS$ (</td>
</tr>
<tr>
<td>$\bar{x}(t) \Rightarrow {Q(R)(t) \mid R \in E}$)</td>
</tr>
<tr>
<td>$\land \not\exists \bar{x}' \exists E' \in RS$ (</td>
</tr>
<tr>
<td>$\forall t' &lt; t \ (\bar{x}'(t') = \bar{x}(t'))$</td>
</tr>
<tr>
<td>$\land \ E &lt; E'$</td>
</tr>
<tr>
<td>$\land \bar{x}'(t) \Rightarrow {Q(R)(t) \mid R \in E'}$)</td>
</tr>
<tr>
<td>$\land \forall d \forall e \forall R \in E$ (</td>
</tr>
<tr>
<td>$\bar{x}(t) \Rightarrow d \land ((d \Rightarrow e) \in Q(R)(t))$</td>
</tr>
<tr>
<td>$\Rightarrow e \subseteq Q(R)(t))$)</td>
</tr>
<tr>
<td>(iv) $Q(R)(t)$ at each $t$ is the smallest set satisfying (i)-(iii)</td>
</tr>
</tbody>
</table>
Example 3: Absence of back propagation

\[ P = ( (\emptyset(\{D,E,F\}), \emptyset), \text{DS}) \]

\[ \text{DS} = \{ \begin{array}{l}
D \iff y = 0, \\
E \iff \Box (y' = 1 \land x' = 0), \\
F \iff \Box (y = 5 \implies x = 1) 
\end{array} \} \]

a. \[ y(t) = t, \quad x(t) = 1 \] satisfies \( D, E, F \) at \( 0 \leq t \).

b. \[ y(t) = t, \quad x(t) = 2 \] satisfies \( D, E, F \) at \( 0 \leq t < 5 \) and \( D, E \) at \( t = 5 \). It again satisfies \( D, E, F \) at \( t \geq 5 \).

c. \[ y(t) = t, \quad x(t) = 2 \quad (t < 5), \quad x(t) = 1 \quad (t \geq 5) \] satisfies \( D, E, F \) at \( 0 \leq t < 5 \) and \( D, F \) at \( t = 5 \). It again satisfies \( D, E, F \) at \( t \geq 5 \).

All of a., b. and c. satisfy local maximality and hence satisfy \( P \).
Example 4: Bouncing Ball, revisited

\[ P = (RS, DS) \]

\[ RS = (\{{\{I,C,B\},\{I,C,F,B\}\},\{{\{I,C,B\} < \{I,C,F,B\}\}}) \]

\[ DS = \{ I \iff ht = 10 \land ht' = 0, \]

\[ \quad C \iff \Box(g = 9.8 \land c = 0.5), \]

\[ \quad F \iff \Box(ht'' = -g), \]

\[ \quad B \iff \Box(ht- = 0 \Rightarrow ht' = -c \times (ht' - )) \}\]

- \(ht\) and \(ht'\) are not differentiable when bouncing
- However, to solve ODEs on \(ht\) and \(ht'\), right continuity of \(ht\) and \(ht'\) at the bouncing must be assumed
- To determine \(ht\) at the bouncing, left continuity of \(ht\) must be assumed as well. (cf. \(ht'\) is determined from \(B\).)

Trajectories with differential constraints should assume both right and left continuity with appropriate priority.
Example 5: Behaviors defined without ODEs

\[ P = (RS, DS) \]
\[ RS = (\{\{A,C\}, \{A,B,C\}\}, \{\{A,C\} \prec \{A,B,C\}\}) \]
\[ DS = \{ A \iff f=0 \land \Box(f' = 1), \]
\[ B \iff \Box(g=0), \]
\[ C \iff \Box(f=5 \Rightarrow \exists a.(a=0 \land \Box(a'=1) \land \Box(a=2 \Rightarrow g=1))) \} \]

- \( g \) is an impulse function that fires at time 7 (= 5+2).
- an example of non-right-continuous functions

\[ \Box(0.9 < a \land a < 1.1) \land \Box(a' = b) \]

- \( a \) is a set of all smooth trajectories with the range (0.9, 1.1). Could be used for specification but not for modeling.
Example 6: Zeno behavior

\[ P = (RS, DS) \]
\[ RS = (\{(I, Pa, B), (I, Pa, F, B)\}, \{(I, Pa, B) \prec (I, Pa, F, B)\}) \]
\[ DS = \{ I \iff ht = 10 \land ht' = 0, \]
\[ Pa \iff \Box(g = 9.8 \land c = 0.5), \]
\[ F \iff \Box(ht'' = -g), \]
\[ B \iff \Box(ht = 0 \Rightarrow ht' = -c \times (ht')) \}\]

- This doesn't define a trajectory \textit{after} the Zeno time.
- A rule for defining the trajectory after Zeno:

\[ \Box(ht = 0 \land ht' = 0 \Rightarrow \Box(ht = 0)) \]

- Checking of the guard condition would require a technique not covered by the current operational semantics.
Execution algorithm and implementation
HyLaGI: A symbolic simulator

- C++ (frontend) and Mathematica (backend), 27kLOC
- KV library\(^1\) for interval computation
- Optimized computation by exploiting the locality of constraints
- webHydLa\(^2\) for visualization

## Rigorous tools for hybrid systems

<table>
<thead>
<tr>
<th>Tool</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acumen</td>
<td>Validated Numerical Simulation</td>
</tr>
<tr>
<td>Flow*</td>
<td>Taylor model + Domain contraction</td>
</tr>
<tr>
<td>dReach/dReal</td>
<td>Interval Constraint Propagation + Bounded Model Checking with Unrolling + SMT Solving</td>
</tr>
<tr>
<td>SpaceEx</td>
<td>Template Polyhedra &amp; Support functions</td>
</tr>
<tr>
<td>KeYmaera &amp; KeYmaera X</td>
<td>Symbolic Theorem Prover based on differential invariants</td>
</tr>
<tr>
<td><strong>HyLaGI</strong></td>
<td>Symbolic + Affine Arithmetic + Interval Newton method</td>
</tr>
</tbody>
</table>
Execution algorithm of HydLa should handle:

1. conditions that start to hold “after” some time point
   - need to compute the greatest lower bound of a time interval
     - $A \iff x=0$.  
     - $B \iff \square (y=1)$.  
     - $C \iff \square ((x'=1 \land (x>3 \Rightarrow y=2)))$.  
     - $A, (B << C)$.  

2. initial values given as intervals
   - could be divided into a subinterval that entails a guard and another that does not entail the guard

3. systems with symbolic parameters
   - needs symbolic computation
For simulation, we need to consider a class of “computable” trajectories.

Computable trajectories: those that have possibly \textit{parametric} equational closed forms

- ODEs without closed-form solutions are to be over-approximated by parametric equational closed forms.
Execution algorithm

each phase updates the maximal consistent set and simulation time $T$

SS (store set) : set of possible stores

failure: choose the next candidate set and redo PP or IP

an element of SS represents a result of execution of PP or IP

compute poset of constraints

compute Point Phase (PP)

compute Interval Phase (IP)

end time? yes no

|SS| = 1

branch of trajectory: nondeterministically choose one element from SS and redo PP or IP

tries the top candidate first

INIT PARAMS FALL BOUNCE

INIT PARAMS BOUNCE
Closure calculation repeatedly checks the antecedents of conditional constraints.

IP computes the next jump time (minimum of the following):
1. a conditional constraint becomes effective
2. a conditional constraint becomes ineffective
3. a ruled-out constraint becomes consistent with effective ones
4. the set of effective constraints becomes inconsistent
Where’s nondeterminism?

- Choice of *maximally* consistent set of rules
- Calculating deductive closure
  - Guard \((g \Rightarrow \cdots)\) may hold or may not hold depending on parameter values
    (e.g., will the thrown ball reach the wall?)
  - We calculate a “strengthened” constraint store for each case
- Finding the next possible jumps time
  - Reason of the next jump may depend on parameter values
    (e.g., will the ball hit the wall or the floor first?)
  - Together with each jump time, calculate a strengthened constraint store which causes that jump first
Example: Bouncing ball with ceiling

- Thrown towards ceiling from some unknown height

HydLa

INIT ⇔ 9 ≤ y ≤ 11 ∧ y' = 10.
FALL ⇔ □(y'' = −10).
BOUNCE ⇔ □(y− = 15 ⇒ y' = −(4/5) * y'−).

INIT, (FALL << BOUNCE).
Symbolic execution of HydLa models

- Use symbolic parameters to handle uncertainties
- Includes ODE solving, Quantifier Elimination (for consistency checking and case splitting), optimization problem (for computing time of discrete change)

Case 1 (fall)
\[ 9 < y(0) < 10 \]
\[ y : 10t - 5t^2 + y(0) \]

Case 2 (touch)
\[ y(0) = 10 \]
\[ y : 10t - 5t^2 + 10 \]

Case 3 (collide)
\[ 10 < y(0) < 11 \]
\[ y : 15 - \sqrt{10y(0)} \ldots \]
In which case can a ball reach here?

Uncertainty in the initial value of $x'$

Bouncing ball on a ground with a hole

$0 \leq x'(0) \leq 20$

$y'(0) = 0$

$y''(t) = -10$

$y'(t) := -\frac{4}{5} \times y'$
Bouncing ball on a ground with a hole

INIT  <=>  y = 10  \land  y' = 0  \land  x = 0  \land  0 \leq x' \leq 20.
FALL  <=>  □(y'' = -10).
BOUNCE  <=>  □(y'' = -10  \lor  (x\leq 7  \lor  x\geq 10)  \land  y'' = 0  \\
            \Rightarrow  y' = -(4/5) \times y'')
XCONST  <=>  □(x'' = 0).
XBOUNCE  <=>  □((x\leq 7  \lor  x\geq 10)  \land  y'' < 0  \Rightarrow  x' = -x'')

INIT, (FALL <= BOUNCE), (XCONST <= XBOUNCE).
ASSERT( ! (y \geq 0  \land  x \geq 10)).

Search when the ball reaches the goal zone
Bouncing ball on a ground with a hole

\[ \text{INIT} \leftrightarrow y = 10 \land y' = 0 \land x = 0 \land 0 \leq x' \leq 20. \]

\[ \text{FALL} \leftrightarrow \Box (y'' = -10). \]

\[ \text{BOUNCE} \leftrightarrow \Box (y^- = -7 \mid (x^- \land 7 \mid x^- \geq 10) \land y^- = 0 \Rightarrow y' = -\frac{4}{5} \cdot y'). \]

\[ \text{XCONST} \leftrightarrow \Box (x'' = 0). \]

\[ \text{XBOUNCE} \leftrightarrow \Box ((x^- = 7 \mid x^- = 10) \land y^- < 0 \Rightarrow x' = -x'). \]

\[ \text{INIT, FALL} \ll \text{BOUNCE}, \text{XCONST} \ll \text{XBOUNCE}. \]

\[ \text{ASSERT}(! (y \geq 0 \land x \geq 10)). \]

Successfully simulated with automatic case analysis
(50 cases including unreachable ones)
(up to 20 seconds, six discrete changes)

Search when the ball reaches the goal zone
Bouncing ball on a ground with a hole (1/9)

\[ x'(0) = [1.36027, 1.40428] \]
Bouncing ball on a ground with a hole (2/9)

\[ x'(0) = [1.82244, 1.90375] \]
Bouncing ball on a ground with a hole (4/9)

\[ x'(0) = [2.643, 2.71964) \]
**Bouncing ball on a ground with a hole (5/9)**

$x'(0) = [2.71964, 4.94975]$
Bouncing ball on a ground with a hole (6/9)

\[ x'(0) = (5.33196, 5.42326) \]
Bouncing ball on a ground with a hole (7/9)

\[ x'(0) = 5.42326 \]

Diagram showing a ball bouncing on a ground with a hole. The diagram includes axes for the horizontal (x) and vertical (y) directions, with labels for the floor, left, right, and bottom boundaries. The ball's position is marked at various points, illustrating its trajectory and interactions with the ground and hole.
Bouncing ball on a ground with a hole (8/9)

\[ x'(0) = (5.42326, 6.56241) \]
Bouncing ball on a ground with a hole (9/9)

\[ x'(0) = [7.07107, 20] \]
Instantaneous events

◆ Hybrid systems handle discrete events
  ● as abstraction of quick physical change (e.g., collision)
  ● to represent computational aspects (e.g., controller)

◆ Superdense time allows multiple events at the same time
  ● \((t, n)\)
    ■ \(t\): real
    ■ \(n = 0, 1, 2, \ldots\): event number at time \(t\)

◆ In our constraint-based framework, what can we do with the standard notion of time?
Modeling behaviors with symbolic perturbation

- Simultaneous collision

[1], Fig.8-9

- Collision + pushing at $1 \leq t \leq 3.5$

[1], Fig.11-12 (equal mass)
[1], Fig.14 (different mass)

- Masses with friction

[1], Fig.15

[1], Fig.28

Solution 1: Form a network of constraints

\[
N := \{n_0 \ldots n_5\}.
\]
\[
F := \{f_0 \ldots f_5\}.
\]
\[
\left[(f_0 = 1 \& n_0 = n \& f = f_5)\right].
\]

\[n = 3.\]
\[
\{ \left[(N[i] > 0 \Rightarrow F[i+1] = F[i] \times N[i] \& N[i+1] = N[i] - 1), \right.
\]
\[
\left. \left[(N[i] \leq 0 \Rightarrow F[i+1] = F[i] \& N[i+1] = N[i])\right] \text{ } | \text{ } i \in \{1..|F|-1\} \right\}.
\]

Solution 2: use \(\exists\)

\[
F(0, y) \iff y = 1.
\]
\[
F(x, y) \& x > 0 \iff \exists z.(y = n \times z \& F(x-1, z))
\]
Cooperation of symbolic and numeric techniques

Discrete change is often hard(er) to compute

Example: water level control

\[
\begin{align*}
\frac{dx_1}{dt} &= -x_1 + 3 \quad (v_1: \text{open}) \\
\frac{dx_1}{dt} &= -x_1 - 2 \quad (v_1: \text{closed}) \\
1 \quad (v_1 \rightarrow \text{close}) \\
-1 \quad (v_1 \rightarrow \text{open}) \\
1.9 \leq x_1(0) \leq 1.9001
\end{align*}
\]

\[
\begin{align*}
x_2(0) &= 1 \\
1 \quad (v_1 \rightarrow \text{close} \& v_2 \rightarrow \text{open}) \\
0 \quad (v_2 \rightarrow \text{close})
\end{align*}
\]

\[
\begin{align*}
\frac{dx_2}{dt} &= x_1 - x_2 - 5 \\
&\quad (v_2: \text{open}) \\
\frac{dx_2}{dt} &= x_1 \quad (v_2: \text{closed})
\end{align*}
\]

- First continuous change
  \[x_2(t) = -\frac{-8 + 7e^t - 2t - t \times x_1(0)}{e^t}\]

- Mathematica cannot symbolically solve
  \[x_2(t) = 0\]

- We need to handle it with interval numerical methods
Interval arithmetic

Arithmetic defined on intervals of reals
  • e.g. \([a, b] + [c, d] = [a + c, b + d]\)
    \([a, b] - [c, d] = [a - d, b - c]\)

Shortcoming: explosion of interval width

Cause 1: Wrapping Effect

\(X := [-1, 1]\)
\(f(x) := x - x\)
\(f(X) = X - X\)
\(= [-1, 1] - [-1, 1]\)
\(= [-2, 2]\)

Solve by handling symbolic parameters
Symbolic vs. numerical methods

**Symbolic**

**Advantage**
Retains parametric info

**Disadvantage**
Growth of size of math formulae

**Numerical**

**Advantage**
Handles vast class of models

**Disadvantage**
Accumulation of errors

*Tradeoff*

---

![Graph](image-url)
Cooperation of symbolic and numeric methods

◆ Use **affine arithmetic (AA)** to approximate complex formulae
  ● to reduce computational cost
  ● while retaining linear terms of parameters

◆ Use **interval Newton method** and **mean-value theorem** to compute discrete change rigorously
  ● to handle systems that are hard to compute symbolically
  ● while retaining linear terms of parameters
Cooperation of symbolic and numeric methods

Compute discrete changes by solving (in)equalities

Over-approximate by affine arithmetic (+ reduction)

Compute continuous changes by solving ODEs

Compute time of events by computing zero crossing of functions

Compute Zero crossing by interval Newton

Refine solution by mean value thm

: symbolic

: cooperation with numerical methods
Affine Arithmetic

- Extended version of Interval Arithmetic
  - Expresses uncertainty in affine form

Affine form

\[
\begin{align*}
X &= x_0 + x_1 \varepsilon_1 + \cdots + x_n \varepsilon_n \\
-1 &\leq \varepsilon_i \leq 1
\end{align*}
\]

- Each \( \varepsilon_i \) represents uncertainty just in the same manner as symbolic parameters in symbolic execution
- Each \( x_i \) (\( i > 0 \)) represents the effect of \( \varepsilon_i \), while \( x_0 \) represents the center

Affine Arithmetic

- Affine forms represent zonotopes, a polygon with parallel opposite edges
- Symbolic parameters $\varepsilon_i$ retain first-order dependencies between uncertain values

\[
X = 5 + \varepsilon_1 - \varepsilon_2 + \varepsilon_3
\]
\[
Y = 5 + \varepsilon_1 + \varepsilon_2 + 0.5\varepsilon_3
\]

\[
X = [2, 8]
\]
\[
Y = [2.5, 7.5]
\]
We use affine arithmetic to over-approximate symbolic formulas

- It reduces computational cost for complex formulas
- Number of preserved parameters can be reduced

**Example**

\[
\begin{align*}
f(x) &:= (x + 1)^2 - 2x \\
X &:= 0 + 0.1 \varepsilon_1 (= [-0.1, 0.1]) \\
f(X) &= (1 + 0.1\varepsilon_1)^2 - 0.2 \varepsilon_1 \\
&= 2(1 + 0.1\varepsilon_1) - 0.995 + 0.005 \varepsilon_2 - 0.2 \varepsilon_1 \\
&= 2 + 0.2\varepsilon_1 - 0.2 \varepsilon_1 - 0.995 + 0.005 \varepsilon_2 \\
&= 1.005 + 0.005\varepsilon_2 (= [1, 1.01])
\end{align*}
\]
Computation of Event Time

◆ Goal: compute the solution of \( f(t, \hat{p}) = 0 \) w.r.t. \( t \) that preserves the linear terms of the parameters \( \hat{p} \)

◆ Assume that the guard is described by a single equation: 
\[
g(\hat{x}) = 0
\]

Step 1. Substitute solution of ODEs into \( g(\hat{x}) \) and obtain \( f(t, \hat{p}) \)

Step 2. Solve \( f(t, \hat{p}) = 0 \) by interval Newton method and obtain solution interval \( T \)

Step 3. Obtain linear over-approximation \( F(t, \hat{p}) \) that encloses \( f(t, \hat{p}) \) in \( T \) using mean value thm

Step 4. Compute zero-crossing of \( F(t, \hat{p}) \) symbolically
Step 1. Substitution of Trajectory

Event time is the positive minimal time satisfying the guard.

Trajectory: \( x = -0.5 + 0.2 t^2 \land y = -0.3 + \sin(3t) + \frac{\varepsilon}{100} \)

Guard: \( g(x, y) = x^2 + y^2 - 1 = 0 \)
Extended version of Newton method

**Features:**

- Computes over-approximated zero-crossing of $f(t, \varepsilon)$
- Converges quadratically
- Guarantees existence and uniqueness of solution

Step 2. Solution of Interval Newton Method

- Narrow enough along the time axis
Step 2. Solution of Interval Newton Method

- Narrow enough along the time axis, but
- Not optimal along the parameter axis
Step 3. Refinement by Mean Value Theorem

Derive **parametrized** solution from solution **interval**

- Compute parametrized over-approximation of \( f(t, \varepsilon) \)

By mean value theorem for multivariate function

\[
[b, a] \subseteq I \Rightarrow h(b) \in h(a) + \nabla h(I) \cdot (b - a)
\]

Within \( I \), \( h(x) \) is surrounded by the steepest slope and the most moderate slope that passes \((a, h(a))\)
Step 3. Refinement by Mean Value Theorem

From \( h(b) \in h(a) + \nabla h(I) \cdot (b - a) \), by replacing \( h(x) \) with \( f(t, \varepsilon) \), we obtain

\[
\begin{align*}
    f(t, \varepsilon) & \in f(T_m, \varepsilon_m) + \frac{\partial f(T, [-1,1])}{\partial t} (t - T_m) + \frac{\partial f(T, [-1,1])}{\partial \varepsilon} (\varepsilon - \varepsilon_m) \\
    &= f(T_m, 0) + \frac{\partial f(T, [-1,1])}{\partial t} (t - T_m) + \frac{\partial f(T, [-1,1])}{\partial \varepsilon} \varepsilon \\
    &=: F(t, \varepsilon)
\end{align*}
\]

\( T_m \) is midpoint of \( T \)

\( \varepsilon_m = 0 \) is midpoint of \( \varepsilon \)

Evaluated to intervals

remaining symbols
Step 3. Refinement by Mean Value Theorem

From \( h(b) \in h(a) + \nabla h(I) \cdot (b - a) \), by replacing \( h(x) \) with \( f(t, \epsilon) \), we obtain

\[
f(t, \epsilon) = f(T_m, \epsilon_m) + \frac{\partial f(T, [-1, 1])}{\partial t} (t - T_m) + \frac{\partial f(T, [-1, 1])}{\partial \epsilon} (\epsilon - \epsilon_m)
\]

\( T_m \) is midpoint of \( T \)

\( \epsilon_m = 0 \) is midpoint of \( \epsilon \)

\[
= f(T_m, 0) + \frac{\partial f(T, [-1, 1])}{\partial t} (t - T_m) + \frac{\partial f(T, [-1, 1])}{\partial \epsilon} \epsilon
\]

\( =: F(t, \epsilon) \)

Evaluated to intervals

Zero-crossing of \( F(t, \epsilon) \) is computed analytically
Zero crossing of $F(t, \varepsilon)$ is

$$t = -\frac{f \partial \varepsilon}{f \partial t} \cdot \varepsilon + T_m - \frac{f(T_m, 0)}{f \partial t}$$

The solution preserves the linear term of $\varepsilon$
If guards are described by inequalities, we compute zero-crossings of each atomic condition.

Guard: \( f_1(t) \geq 0 \land f_2(t) \geq 0 \land f_3(t) \geq 0 \)

The earliest time when the whole guard is satisfied.
Evaluation by Examples

Water Level Control

- Compared with naive interval arithmetic
- Preserve 6 symbolic parameters (4 for water level + derivatives, time, additional)
Error width converged in the proposed method
Execution time is longer than naive interval arithmetic, but did not explode.
Bouncing Ball on Sine Wave

- Trajectory of particle
- Sine-shaped floor

- Compared with naive interval arithmetic
- Preserved \{5, 9, 13\} parameters
Error width of Bouncing Ball

- Newton & Interval
- Affine & Newton & Mean $e^5$
- Affine & Newton & Mean $e^9$
- Affine & Newton & Mean $e^{13}$

Compared with naive interval arithmetic
Execution time of Bouncing Ball

![Graph showing the execution time of Bouncing Ball for different methods. The x-axis represents the step number, and the y-axis represents execution time (s). The graph includes lines for Newton & Interval, Affine & Newton & Mean e5, Affine & Newton & Mean e9, and Affine & Newton & Mean e13.]

- **Tradeoff between error width and execution time**
Thanks for the attention!